23/24 AN2DL Challenge 1

Giacomo Ballabio - 10769576
Christian Biffi - 10787158
Alberto Cavallotti - 10721275

Deep Sheets

1 Introduction

The objective of this project was to classify two types of plant images: healthy and unhealthy. We were provided with a dataset comprising 5100 images, and our task was to design a network for optimal classification on an unknown test set.

2 Dataset analysis

Initially, we examined the dataset to understand its composition and eliminate ”outliers”. While we observed a slight imbalance between the two classes, we chose not to apply transformations for balance due to the minimal difference.

We employed one-hot encoding for labels and initially split the dataset into an 80/20 ratio for training and validation. However, during experimentation, we favored a 90/10 ratio to enhance training with the limited dataset.

3 First approach

We started by implementing a basic hand-crafted CNN made of 4 convolutional layer inspired by the LeNet architecture [1], followed by a FC network and a softmax as the activation function. We experimented with various configurations, including adjusting filter numbers, kernel sizes, and FC layer composition. We also tried to replace the FC network with a Global Average Pooling Layer and the output layer made of two neurons, as we have two classes. This last approach gave us the best results, so we continued to use this approach for the following experiments. Since the network was not performing well, clearly overfitting, we implemented the batch normalization after each convolutional layer and this helped in the final result. In parallel we started to test and use some Data Augmentation techniques such as Flip and CentralCrop.

In conclusion our best test locally, using also data augmentation, scored an accuracy of 88%, but when uploaded on the competition only scored a 64% accuracy on the actual test set, so we started to move on to transfer learning method.

4 Transfer Learning and Finetuning

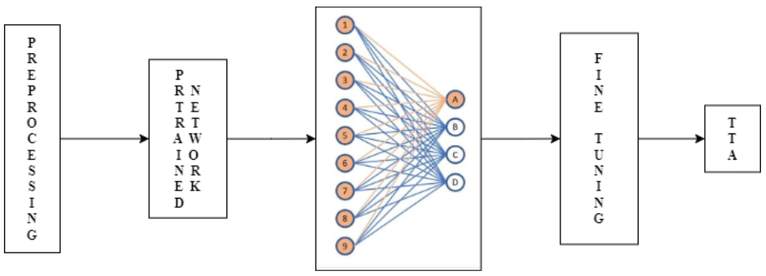
As seen during lecture we started to use the transfer learning approach with pre-trained networks available in Keras Application [2]. We tried many of the architecture available in order to understand which of them adapted better for our case, to name a few we tried Xception [3], MobileNetV2 [4], NASNetMobile [5], ResNet [6] with various dimensions, but the one that was giving us the bests result was the EfficientNetV2

[7] architecture.

From the various version of the EfficientNetV2, we tried all of them apart from the EfficientNetV2 L, in order to understand which of them was the most suitable for our problem. After all the test we found out that EfficientNetV2 S was the one performing better, so we started to use it consistently for our experiments.

Since at a certain point we felt like having hit a sort of upperbound using EfficientNetV2 and we couldn't find a way to improve even more, we started to approach ConvNeXt [8] architecture and we noticed right away that it had great potentiality. In particular the best variant was the ConvNeXt Base model.

With these two last models we started to work on the tuning of the hyperparameters and the FC layers following this approach:



4.1 Preprocessing

We started to apply several Data Augmentation techniques, as already tested in handcrafted CNN, to understand which of the transformation available as Keras layers were performing better for our problem. In particular we tried random flip, various degree of rotation, translation, central crop, contrast and brightness, but in the end the ones from which we obtained the best scores were random flip, central crop and contrast.

Even with these different tests, our experiments were not gaining a lot in accuracy, the real first turning point were when we started to upscale our images to match the dimension used for the train done on ImageNet dataset by the pre-trained model that we have used. In particular for EfficientNetV2 we upscaled the images to $(128, 128, 3)$, which is the minimum dimension recommended. While for ConvNeXt we resized the images to $(224, 224, 3)$. This approach started to give us much higher accuracies (around 88%) on the test set.

The second turning point was when we started to use the more complex transformation like MixUp and CutMix in combination with RandAugment taken from KerasCV [9]. In particular CutMix wasn't working really well, but MixUp was the one that permitted us to gain another 3% on the test set, with the best result of 92% accuracy.

4.2 Fully Connected Network

Starting from the knowledge gained with the tests done on the handcrafted CNN, we initially started with a setup made of a Global Averaging Pooling Layer after which we tested to add both one FC layer and two FC layers. We made different tests on the number of neurons and also added Batch Normalization and Dropout layer in order to regularize data and avoid overfitting.

The combination that was giving us the best results in general was with two FC layers made of 512 neuron and 256 neurons. After this was always present the output layer with two neurons to classify the images.

Network	Dense layer	DroupOut Value	Accuracy
ConvNeXt	512+256	0.16667	93%
EfficientNet	512+256	0.16667	93%
ConvNeXt	512	0.16667	93%
EfficientNet	512+256	0.7	91%
ConvNeXt	512	0.25	89%
EfficientNet	512+256	0.8	89%

Table 1: Table of our best networks

4.3 Fine Tuning

After the train of the FC network with the pre-trained model freezed, we applied finetuning in order to adapt even more the features extracted by the pre-trained network on our dataset. We made experiments trying to unfreeze different numbers of layers ("blocks" in case of the EfficientNetV2 as suggested in the documentation [10]) but the best results were with the entire pre-trained network unfreezed (this was not possible with ConvNeXt Base since it wouldn't fit in GPU memory). Our standard setup was using 30 epochs for finetuning with checkpoint saving for the best epoch based on `val_loss`.

4.4 Test Time Augmentation

Another important thing that made us gain another 1% on the test set was the application of Test Time Augmentation [11]. We tried again many of the available image transformations to understand which of the combinations were performing the best. We tried flipping, contrast, brightness, gaussian noise and rotation to name a few. After all the best ones that we added to our *model.py* were flipping, rotation, central crop, brightness, contrast and translation.

4.5 Other notes

In all the training made for our networks we used both early stopping and learning rate reduction, the former to avoid overfitting and the latter to make the network learn better.
All the accuracies mentioned in the previous sections references to the development phase.

5 Conclusion

This was a very fun challenge that gave us the possibilities to have a more practical approach to what we were studying even if it forced us to stare at numbers flowing on the screen for days because of our hopes to have a better score; it felt like being in The Matrix [12].
Our best results in the developing phase was 93% that we reached with both EfficientNetV2 S and ConvNeXt Base. Probably with some more time, and/or an earlier approach to the ConvNeXt architecture we would have increased even more our final top accuracy.

Contribution

We equally contributed to the search and development of new models and techniques both explained during the lectures and never seen ones. Most importantly everyone contributed for the training with all the resource available and accounts divided between Google Colab and Kaggle.

References

[1] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[2] François Chollet et al. *Keras*. https://keras.io/api/applications/. 2015.

[3] François Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: *CoRR* abs/1610.02357 (2016). arXiv: 1610.02357. URL: http://arxiv.org/abs/1610.02357.

[4] Mark Sandler et al. "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation". In: *CoRR* abs/1801.04381 (2018). arXiv: 1801.04381. URL: http://arxiv.org/abs/1801.04381.

[5] Barret Zoph et al. "Learning Transferable Architectures for Scalable Image Recognition". In: *CoRR* abs/1707.07012 (2017). arXiv: 1707.07012. URL: http://arxiv.org/abs/1707.07012.

[6] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[7] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *CoRR* abs/1905.11946 (2019). arXiv: 1905.11946. URL: http://arxiv.org/abs/1905.11946.

[8] Zhuang Liu et al. "A ConvNet for the 2020s". In: *CoRR* abs/2201.03545 (2022). arXiv: 2201.03545. URL: https://arxiv.org/abs/2201.03545.

[9] lukewood. *CutMix, MixUp, and RandAugment image augmentation with KerasCV*. https://keras.io/guides/keras_cv/cut_mix_mix_up_and_rand_augment/. 2022.

[10] Yixing Fu. *Image classification via fine-tuning with EfficientNet*. https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/. 2023.

[11] Step Up AI. *How to Correctly Use Test-Time Data Augmentation to Improve Predictions*. 2020. URL: https://stepup.ai/test_time_data_augmentation/.

[12] Wikipedia contributors. *The Matrix — Wikipedia, The Free Encyclopedia*. 2023. URL: https://en.wikipedia.org/w/index.php?title=The_Matrix&oldid=1185193182.

[13] Ross Wightman, Hugo Touvron, and Hervé Jégou. "ResNet strikes back: An improved training procedure in timm". In: *CoRR* abs/2110.00476 (2021). arXiv: 2110.00476. URL: https://arxiv.org/abs/2110.00476.

[14] Sangdoo Yun et al. "CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features". In: *CoRR* abs/1905.04899 (2019). arXiv: 1905.04899. URL: http://arxiv.org/abs/1905.04899.

[15] Hongyi Zhang et al. "mixup: Beyond Empirical Risk Minimization". In: *CoRR* abs/1710.09412 (2017). arXiv: 1710.09412. URL: http://arxiv.org/abs/1710.09412.